Specification and Analysis of Open-Ended Systems with CARMA

Specification and analysis of open-ended systems with CARMA*

Jane Hillston[1] and Michele Loreti[2,3]

[1] Laboratory for Foundations of Computer Science, University of Edinburgh
[2] Dipartimento di Statistica, Informatica, Applicazioni "G. Parenti", Università di Firenze
[3] IMT, Lucca

Abstract. CARMA is a new language recently defined to support quantified specification and analysis of collective adaptive systems. It is a stochastic process algebra equipped with linguistic constructs specifically developed for modelling and programming systems that can operate in open-ended and unpredictable environments. This class of systems is typically composed of a huge number of interacting agents that dynamically adjust and combine their behaviour to achieve specific goals. A CARMA model, termed a "collective", consists of a set of components, each of which exhibits a set of attributes. To model dynamic aggregations, which are sometimes referred to as "ensembles", CARMA provides communication primitives based on predicates over the exhibited attributes. These predicates are used to select the participants in a communication. Two communication mechanisms are provided in the CARMA language: multicast-based and unicast-based. A key feature of CARMA is the explicit representation of the environment in which processes interact, allowing rapid testing of a system under different open world scenarios. The environment in CARMA models can evolve at runtime, due to the feedback from the system, and it further modulates the interaction between components, by shaping rates and interaction probabilities.

1 Introduction

Collective adaptive systems (CAS), comprised on multiple agents working in collaboration, competition or a combination of both, are predicted to form the underlying infrastructure for the next generation of software systems and services. Their transparent and pervasive nature makes it imperative that the behaviour of such systems should be thoroughly explored and evaluated at design time and prior to deployment. In this paper we present a novel modelling language which has been developed to capture the behaviour of such systems and support their analysis through a variety of techniques.

CARMA (Collective Adaptive Resource-sharing Markovian Agents) is a stochastic process algebra-based language which means that it represents systems as interacting agents which undertake actions; actions are assumed to have a duration which is represented as a random variable, governed by an exponential distribution. The quantitative information represented by action durations, extends the reasoning power of the language beyond functional analysis to assess the correct behaviour of the system, to non-functional properties such as performance, availability and dependability. Such analysis

* This work is partially supported by the EU project QUANTICOL, 600708.

supports quantitative prediction and checking of system behaviour, including ensuring that resources will be efficiently and equitably shared — an important and strongly desirable feature of infrastructures. Moreover, the underlying interleaving Markovian semantics of CARMAsupports a form of *local synchrony* as discussed in [1]. Independent actions proceed at their own pace, and this autonomy is lost only when agents are constrained to act together through an explicit interaction.

The compositional nature of the language allows components capturing the behaviour of agent types to be defined and then combined to build up the behaviour of the system. The interaction between agents is represented by explicit communication, both multicast-based and unicast-based, and through implicit communication. Moreover a distinguishing feature of CARMA is the inclusion of an *environment*, represented as a distinct element of the model, which captures the operating principles of the world in which the agents exist. This environmental influence includes controlling the rate and effectiveness of communication between agents. Moreover, the environment itself can evolve during the lifetime of the system, reflecting the adaptive operating conditions of the system. The environment also has a state, which can be used to represent implicit asynchronous communication between agents, analogous to pheromone trails in ant systems. This is broadly in-keeping with the architecture proposed by Weyns *et al.* proposed in [2].

In this paper we present an overview of the language constructs of CARMA, including the representation and role of the environment. The style of presentation is intended to be intuitive and the interested reader can find the formal semantics of CARMA in [3]. Specifically, we illustrate the features of the language in a running example, a *Smart Taxi System*. In this system we consider a set of *taxis* operating in a city, providing service to *users*; both taxis and users are agents, or components within our system. In order to manage the system and allocate user requests among the operating taxis, our city is subdivided into a number of patches arranged in a grid over the geography of the city. The users arrive randomly in different patches, at a rate that depends on the specific time of day. After arrival, a user makes a call for a taxi and then waits in that patch until they successfully engage a taxi and move to another randomly chosen patch. Unengaged taxis move about the city, influenced by the calls made by users.

The rest of the paper is organised as follows. In the next section we give a brief overview of related modelling techniques for CAS. Section 3 gives a detailed account of the CARMA language and demonstrates its use to model the Smart Taxi System. Section 4 presents the result of quantitative analysis of the Smart Taxi System, illustrating the types of measures that can be derived from CARMA models. Previous work on modelling the stochastic dynamic behaviour of a system under the influence of an environment is discussed in Section 5, and we conclude in Section 6.

2 Related work

Stochastic process algebras, such as PEPA [4], MTIPP [5], EMPA [6], Stochastic π-Calculus [7], Bio-PEPA [8], MODEST [9] and others [10, 11], have been available to support quantitative analysis of systems for approximately two decades. Whilst some offer support for large populations of agents, they have not been designed with CAS in

mind, and typically only support synchronous unicast communication. In recent years there have been languages targeted towards systems consisting of populations, collectives or ensembles of agents [12–14], which feature attribute-based communication and explicit representation of locations and the development of CARMA has been informed by these.

SCEL [12] (Software Component Ensemble Language), has been designed to support the programming of autonomic computing systems. The language distinguishes between *autonomic components* representing the collective members, and *autonomic-component ensembles* representing collectives. Each component is equipped with an interface, consisting of a collection of attributes, describing different features of components. There is an underlying knowledge base associated with each component and rich forms of reasoning on attributes are supported. Attributes from the knowledge base are used by components to dynamically form ensembles and to select partners for interaction. The stochastic variant of SCEL, called StocS [15], has been used to investigate a number of different stochastic and probabilistic semantics. Moreover, SCEL has inspired the development of the core calculus AbC [16] that focuses on a minimal set of primitives that defines attribute-based communication without the rich underlying knowledge base, and investigates their impact. Communication among components takes place in a broadcast fashion, with the characteristic that only components satisfying predicates over specific attributes receive the sent messages, provided that they are willing to do so.

PALOMA [13] is a process algebra that takes as starting point a model based on located Markovian agents each of which is parameterised by a location, which can be regarded as an attribute of the agent. The ability of agents to communicate depends on their location, through a *perception function*, which can be used to define the range of a communication. Examples include *local* – agents must be co-located, *all* – communication is global, or use of predicates. This can be regarded as an example of a more general class of attribute-based communication mechanisms. Both multicast and unicast communication are supported [17], but in both cases only agents who enable the appropriate reception action have the ability to receive the message. The scope of communication is thus adjusted according to the perception function. PALOMA is supported by an individual-based Markovian semantics, a population-based Markovian semantics and a continuous semantics in the style of [18].

The attributed pi calculus [14] is an extension of the pi calculus [19] that supports attribute-based communication, and was designed primarily with biological applications in mind. As in the languages discussed above, processes may have attributes and these are used to select partners for interaction, but note that communication is strictly synchronisation-based and binary. The language is equipped with both a deterministic and a Markovian semantics, and in the Markovian case the rates may depend on the attribute values of the processes involved. The possible attribute values are defined by a language $\mathscr{L}$, and the definition of the attributed pi calculus is parameterised by $\mathscr{L}$. The language $\mathscr{L}$ also defines the possible rates and constraints that can be applied to attributes, giving the possibility to capture diverse behaviours within the framework when rates and probabilities of interaction are all dependent only on local behaviour and knowledge.

CARMA supports both unicast and broadcast communication, providing locally synchronous, but globally asynchronous communication. This richness is important to enable the spatially distributed nature of CAS, where agents may have only local awareness of the system, yet the design objectives and adaptation goals are often expressed in terms of global behaviour. Representing these rich patterns of communication in classical process algebras or traditional stochastic process algebras would be difficult, and would require the introduction of additional model components to represent buffers, queues and other communication structures. Moreover the inclusion of a distinct environment, allows many possibilities including rates of behaviour based on the global state, or observed data from a system.

Modelling frameworks and languages have also been developed within the multi-agent community, but generally with a focus on specification, qualitative or functional analysis, rather than quantitative assessment of the modelled system e.g. [20–23]. For example, Helleboogh *et al.* consider the requirements for modelling multi-agent systems, with a particular emphasis on the modelling of physical applications via simulation [20]. As with CARMA, the authors make a distinction between the *simulated environment* and the *simulating environment*, which is transparent to the modeller. However, in the work presented in [20], the distinction between the simulated environments and the agents within it is not strong (cf. the framework of Weyns *et al*. [2]). In CARMA we prefer to have a clear separation of concern which allows the same collective of agents to be easily considered in the context of different environments. Moreover, whereas in CARMA agents explicitly interact, in Helleboogh *et al.*'s approach the behaviour of agents is specified in a declarative way with the environment playing a coordinating role in the evolution of agents.

3 CARMA in a nutshell.

In this section we present the CARMA language and its specific features. To simplify the presentation, and to help the reader to appreciate CARMA features, we will consider a simple running scenario. The latter is a *Smart Taxi System* used to coordinate the activities of a group of taxis in a city. In our scenario we assume that the city is subdivided in patches forming a grid. Two kinds of agents populate the system: *taxis* and *users*. Each taxi stay can either stay in a patch, waiting for user requests, or move to another patch. Users randomly arrive at different patches with a rate that depends on the specific time of day. After arrival, a user waits for a taxi and then moves to another patch.

The *smart taxi scenario* represents well the typical scenarios modelled with CARMA. Indeed, a CARMA system consists of a *collective* (N) operating in an *environment* ($\mathscr{E}$). The collective consists of a set of components. It models the behavioural part of a system and is used to describe a set of interacting *agents* that cooperate to achieve a given set of tasks. The environment models all those aspects which are intrinsic to the context where the agents under consideration are operating. The environment also mediates agent interactions.

Example 1. In our running example the collective N will be used to model the behaviour of *taxis* and *users*, while the environment will be used to model the city context where these agents operate.

In CARMA collectives are defined in a style that is borrowed from process algebras. We let COL be the set of collectives N which are generated by the following grammar:

$$N \ ::= \ C \ \mid \ N \parallel N \qquad C \ ::= \ \mathbf{0} \ \mid \ (P, \gamma)$$

where C denotes a *component* while $N \parallel N$ represents the parallel composition of two collectives.

A component C can be either the *inactive component*, which is denoted by $\mathbf{0}$, or a term of the form (P, γ), where P is a *process* and γ is a *store*. A term (P, γ) models an *agent* operating in the system under consideration: the process P represents the agent's behaviour whereas the store γ models its *knowledge*. The latter is a function which maps *attribute names* to *basic values*. In the rest of this paper we let:

- ATTR be the set of *attribute names* a, a', a_1,..., b, b', b_1,...;
- VAL be the set of *basic values* v, v', v_1,...;
- Γ be the set of *stores* $\gamma, \gamma_1, \gamma', \ldots$, i.e. functions from ATTR to VAL, of the following form:

$$\{a_0 = v_0, \ldots, a_n = v_n\}$$

Example 2. To model our *smart taxi system* in CARMA we need two kinds of components. One for each of the two groups of agents involved in the system, i.e. *taxis* and *users*. Both the kind of components use the local store to publish the relevant data that will be used to represent the state of the agent.

The local store of components associated to taxis contains the following attributes:

- *loc*: identifies current taxi location;
- *occupancy*: ranging in $\{0,1\}$ describes if a taxi is free ($occupied = 0$) or engaged ($occupied = 1$);
- $destination$: if occupied, this attribute indicates the destination of taxi journey.

Similarly, the local store of components associated with users contains the following attributes:

- *loc*: identifies user location;
- $destination$: indicates user destination.

The behaviour of a component is specified via a process P. We let PROC be the set of processes P, Q,... defined by the following grammar:

$$P, Q ::= \mathbf{nil} \ \mid \ A[X] \ \mid \ P \mid Q$$

where **nil** denotes the inactive process, $A[X]$ denotes *process automaton* A in state X, while $P \mid Q$ denotes the behaviour obtained from the concurrent execution of P and Q. Each process automaton, and its states, are defined as follow:

$$\begin{array}{l}
\mathsf{process}\ A = \\
\qquad X_0 : [\pi_i^0] act_0^0 . R_0^0 + \cdots + [\pi_i^{k_0}] act_0^{k_0} . R_0^{k_0} \\
\qquad \vdots \qquad\quad \vdots \qquad\qquad\qquad \vdots \\
\qquad X_n : [\pi_i^n] act_n^n . R_n^n + \cdots + [\pi_n^{k_n}] act_n^{k_n} . R_n^{k_n} \\
\mathsf{endprocess}
\end{array}$$

Above, π_i^j is a *guard*, act_i^j are CARMA actions, while R_i^j can be a state in $\{X_0,\ldots,X_n\}$, the inactive process **nil** or the term **kill**. A process $A[X_i]$, that is active in a component with store γ, can execute any of the actions $act_i^{k_i}$ such that π_i^j is satisfied by γ. When one enabled action act_i^j is executed, process $A[X_i]$ terminates, when $R_i^j = \mathbf{nil}$, evolves to $A[X_l]$, when $R_i^j = X_l$ and *destroys* the enclosing component when $R_i^j = \mathbf{kill}$.

In CARMA processes can perform four types of actions: *broadcast output* ($\alpha^\star[\pi]\langle\overrightarrow{e}\rangle\sigma$), *broadcast input* ($\alpha^\star[\pi](\overrightarrow{x})\sigma$), *output* ($\alpha[\pi]\langle\overrightarrow{e}\rangle\sigma$), and *input* ($\alpha[\pi](\overrightarrow{x})\sigma$), where:

- α is an *action type* in the set of action types ACTTYPE;
- π is an *predicate*;
- x is a *variable* in the set of variables VAR;
- $\overrightarrow{\cdot}$ indicates a sequence of elements;
- σ is an *update* of the form:

$$\{\mathsf{a}_0 \leftarrow re_0, \cdots, \mathsf{a}_k \leftarrow re_k\}$$

where all the attributes a_i are distinct and *re* is a *random expression* defined by the following syntax:

$$re ::= \mathsf{now} \mid v \mid x \mid \mathsf{this}.a \mid \delta_{\mathrm{VAL}} \mid re\ b_{op}\ re \mid u_{op}\ re \mid f(re_0,\ldots,re_n)$$

where now is used to refer to current system time, δ_{VAL} is a probability distribution over VAL[4], b_{op} and u_{op} are *binary* and *unary* operators, while f denotes an n-ary function over expressions.
In the rest of this paper we will say that an expression *er* is *ground* if it does not contain any variable, while it is *deterministic* if no probability expression δ_{VAL} occurs in *re*. Moreover, we will use *e* to denote a *deterministic expression*.

The admissible communication partners in each of these actions are identified by the predicate π. This is a predicate on *attribute names*. Note that, in a component (P,γ) the store γ regulates the behaviour of P. Primarily, γ is used to evaluate the predicate associated with an action in order to filter the possible synchronisations involving process P. In addition, γ is also used as one of the parameters for computing the actual rate of actions performed by P. The process P can change γ immediately after the execution of an action. This change is brought about by the *update* σ. The update is a function that when given a store γ returns a probability distribution over Γ which expresses the possible evolutions of the store after the action execution.

The *broadcast output* $\alpha^\star[\pi]\langle\overrightarrow{e}\rangle\sigma$ models the execution of an action α that spreads the values resulting from the evaluation of expressions $\overrightarrow{e}$ in the local store γ. This message can be potentially received by any process located at components whose store satisfies predicate π. This predicate may contain references to attribute names that have to be evaluated under the local store. These references are prefixed by the special name this. For instance, if loc is the attribute used to store the position of a component, action

$$\alpha^\star[\mathsf{distance}(\mathsf{this.loc},\mathsf{loc}) \leq L]\langle\overrightarrow{v}\rangle\sigma$$

[4] For any denumerable set X, we let $Dist(X)$ denote the set of probability distributions over X while δ_X is a generic element in $Dist(X)$

potentially involves all the components located at a distance that is less than or equal to a given threshold L. The *broadcast output* is non-blocking. The action is executed even if no process is able to receive the values which are sent. Immediately after the execution of an action, the update σ is used to compute the (possible) *effects* of the performed action on the store of the hosting component where the output is performed.

To receive a broadcast message, a process executes a *broadcast input* of the form $\alpha^\star[\pi](\overrightarrow{x})\sigma$. This action is used to receive a tuple of values $\overrightarrow{v}$ sent with an action α from a component whose store satisfies the predicate $\pi[\overrightarrow{v}/\overrightarrow{x}]$. The transmitted values can be part of the predicate π. For instance, $\alpha^\star[x > 5](x)\sigma$ can be used to receive a value that is greater than 5.

The other two kinds of action, namely *output* and *input*, are similar. However, differently from broadcasts described above, these actions realise a *point-to-point* interaction. The *output* operation is blocking; in contrast to the non-blocking broadcast output.

Example 3. We are now ready to describe the behaviour of *taxis* and *users*. Behaviour of a user is modelled via process *User* defined below:

$$
\begin{array}{l}
\mathsf{process}\ \mathit{User} = \\
\quad W : \mathsf{call}^\star[\top]\langle \mathsf{this.loc}\rangle.W \\
\qquad + \\
\qquad \mathsf{take}[\mathsf{loc} == \mathsf{this.loc}]\langle \mathsf{this.dest}\rangle.\mathbf{kill} \\
\mathsf{endprocess}
\end{array}
$$

Process *User* has only one state, W, that can either *call* or *take* a taxi. To call a taxi, a $\mathit{User}[W]$ executes a *broadcast output* over call to all taxis. A *unicast output* over take is executed to take a taxi. This action is used to send user destination (this.dest) to a taxi that shares the same location as the user. To identify the target of this action, prediate (loc == this.loc) is used. The latter is satisfied only by those components that have attribute loc equal to this.loc. Here prefix this is used to refer to actual values of local attributes. After this action, user disappears (he/she enters the taxi).

The behaviour of a taxi is described via process automaton *Taxi*:

$$
\begin{array}{l}
\mathsf{process}\ \mathit{Taxi} = \\
\quad F : \mathsf{take}[\top](d)\{\mathsf{dest} \leftarrow d, \mathsf{occupied} \leftarrow 1\}.G \\
\qquad + \\
\qquad \mathsf{call}^\star[\mathsf{this.loc} \neq \mathsf{loc}](d)\{\mathsf{dest} \leftarrow d\}.G \\
\quad G : \mathsf{move}^\star[\bot]\langle\circ\rangle\{\mathsf{loc} \leftarrow \mathsf{dest}, \mathsf{dest} \leftarrow \bot, \mathsf{occupied} \leftarrow 0\}.F \\
\mathsf{endprocess}
\end{array}
$$

Process *Taxi* has two state: F and G. When in state F the taxi is *available* and can either take a user in the current location or receive a *call* from another patch. In the first case an input via take is performed and the user destination is received. In the second case, the location where a user is waiting for a taxi is received. In both the cases, the received location is used, after action execution, to update taxi destination (dest $\leftarrow d$). However, after take input, the taxi also records that it is occupied (occupied $\leftarrow 1$).

When in state G, process *Taxi* just executes a *spontaneous* broadcast over action move. Indeed, no process can receive this message since predicate $\bot$ is used. This

models taxi movements. After the movement the taxi position is updated and the taxi is ready to take users in the new location.

To model the arrival of new users, the following process automaton is used:

$$\begin{array}{l}\mathsf{process}\ \mathit{Arrivals} = \\ \qquad A : \mathsf{arrival}^{\star}[\bot]\langle\circ\rangle.A \\ \mathsf{endprocess}\end{array}$$

Process *Arrivals* has a single state A where spontaneous action arrival is executed. This process is executed in a separated component where attribute loc indicates the location where users arrive. The precise role of this process will be clear in a few paragraphs when the environment will be described.

CARMA collectives operate in an environment $\mathscr{E}$. This environment is used to model the intrinsic rules that govern, for instance, the physical context where our system is situated. An environment consists of two elements: a *global store* γ_g, that models the overall state of the system, and an *evolution rule* ρ. The latter consists of three functions that are used to express the probability to receive a message by a component, compute the execution rate of an action, determine the updates to the environment on both the global store and the collective. Syntax of an environment $\mathscr{E}$ is the following:

$$\begin{array}{l}\mathsf{environment}\ \ \gamma\ \ [\\ \qquad \mu_p \\ \qquad \mu_r \\ \qquad \mu_u \\ ]\end{array}$$

Element μ_p is used to compute the probability to receive a message. Syntax of μ_p is the following:

$$\begin{array}{l}\mathsf{prob}\{ \\ \qquad \pi_0, \alpha_0 \to e_0 \\ \qquad \vdots \\ \qquad \pi_{k_p}, \alpha_{k_p} \to e_{k_p} \\ \qquad \mathsf{default}\ \ e \\ \}\end{array}$$

In the above, each π_i is a boolean expression over the stores of the two interacting components, i.e. the sender and the receiver, and while α_i denotes the action used to interact. In π_i attributes of sender and receiver attributes are referred to using $\mathsf{sender}.a$ and $\mathsf{receiver}.a$, while the values published in the global store are referenced by using $\mathsf{global}.a$. A receiver with store γ_1 receives a message sent by a component with store γ_2 under global store γ with probability p if there exists an index i such that:

$$\gamma_1, \gamma_2, \gamma \models \pi_i \qquad \text{and} \qquad [\![e_i]\!]_{\gamma_1,\gamma_2,\gamma} = p$$

or, if none of the π_is are satisfied, $p = [\![e]\!]_{\gamma_1,\gamma_2,\gamma}$. This probability value may depend on the number of components in a given state. For this reason in the environment the syntax of expressions is extended by considering terms of the form:

$$re ::= \cdots \mid \#(\Pi, \pi)$$

where $\#(\Pi, \pi)$ denotes the number of components in the system satisfying predicate π where a process of the form Π is executed. In turn, Π is a pattern of the following form:

$$\Pi ::= \star \mid A[\star] \mid A[X] \mid \Pi|\Pi$$

Example 4. One can use $\#(Taxi[F], \mathsf{loc} == \ell)$ to count the number of available taxis at patch ℓ. This expression can be used as follows:

$$\begin{array}{l}\mathsf{prob}\{\\ \quad \top, \mathsf{take} \rightarrow \frac{1}{\#(Taxi[F], \mathsf{loc}==\mathsf{sender.loc})}\\ \quad \top, \mathsf{call}^\star \rightarrow p_{lost}\\ \quad \mathsf{default}\ \ 1\\ \}\end{array}$$

Above, we say that each taxi receives a user request with a probability that depends on the number of taxis in the patch. Moreover, $\mathsf{call}^\star$ can be missed with a probability p_{lost}. All the other interactions occur with probability 1.

Function μ_r is similar and it is used to compute the rate of an unicast/broadcast output. This represents a function taking as parameter the local store of the component performing the action and the action type used. Note that the environment can disable the execution of a given action. This happens when the function μ_r (resp. μ_p) returns the value 0. Syntax of μ_r is the following:

$$\begin{array}{l}\mathsf{rate}\{\\ \quad \pi_0, \alpha_0 \rightarrow e_0\\ \quad \vdots\\ \quad \pi_{k_r}, \alpha_{k_r} \rightarrow e_{k_r}\\ \quad \mathsf{default}\ \ e\\ \}\end{array}$$

Differently from μ_p, in μ_r guards π_i are evaluated by considering only the attributes defined in the store of the component performing the action, referenced as $\mathsf{sender}.a$, or in the global store, whose elements are accessed via $\mathsf{global}.a$.

Example 5. In our example rate can be defined as follow:

$$\begin{array}{l}\mathsf{rate}\{\\ \quad \top, \mathsf{take} \rightarrow \lambda_t\\ \quad \top, \mathsf{call}^\star \rightarrow \lambda_c\\ \quad \top, \mathsf{move}^\star \rightarrow mtime(\mathsf{now}, \mathsf{sender.loc}, \mathsf{sender.dest})\\ \quad \top, \mathsf{arrival}^\star \rightarrow atime(\mathsf{now}, \mathsf{sender.loc})\\ \quad \mathsf{default}\ \ 0\\ \}\end{array}$$

We say that actions take and $\mathsf{call}^\star$ are executed at a constant rate; the rate of a taxi movement is a function of actual time (now) and of starting location and final destination. Rate of user arrivals depends on current time now and on location loc.

Finally, the function μ_u is used to update the global store and to install a new collective in the system. Syntax of μ_u is:

$$\begin{array}{l}\mathsf{update}\{\\ \quad \pi_0, \alpha_0 \to \sigma_0, N_0\\ \quad \vdots\\ \quad \pi_{k_u}, \alpha_{k_u} \to \sigma_{k_u}, N_{k_u}\\ \}\end{array}$$

Like for μ_r, guards in function μ_u are evaluated on the store of the component performing the action and on global store. However, the result is a pair (σ_i, N_i). Within this pair, σ_i identifies the update on the global store whereas N_i is a new collective installed in the system. If none of the guards are satisfied, or the performed action is not listed, the global store is not changed and no new collective is instantiated. In both cases, the collective generating the transition remains in operation. This function is particularly useful for modelling the arrival of new agents into a system.

Example 6. In our scenario function update is used to model the arrival of new users and it is defined as follows:

$$\begin{array}{l}\mathsf{update}\{\\ \quad \top, \mathsf{arrival}^\star \to \{\ \}, (User[W], \{\mathsf{loc} = \mathsf{sender.loc}, \mathsf{dest} = destLoc(\mathsf{now}, \mathsf{sender.loc})\)\\ \}\end{array}$$

When action $\mathsf{arrival}^\star$ is performed a component associated with a new user is created in the same location as the sender (see Example 3). The destination of the new user will be determined by function *destLoc* that takes current system time and starting location and returns a probability distribution on locations.

To extract data from a system, a CARMA specifications also contains a set of *measures*. Each measure is defined as:

$$\mathsf{measure}\ m\ = e;$$

Example 7. In our scenario, we could be interested in measuring the number of waiting users at a given location. These measures can be declared as:

$$\mathsf{measure}\ \mathsf{waitingAt}_\ell\ = \#(User[\star], \mathsf{loc} == \ell)$$

When the system is simulated a time-ordered sequence of values is generated for each measure specified.

4 The Smart Taxi System: Simulation and Analysis

In this section we present the Smart Taxi System in its entirety and demonstrate the quantitative analysis which can be undertaken on a CARMA model.

In the previous section we have shown how in CARMA a system specification in fact consists of two parts: a collective and an environment. One of the main advantages

$//Component\ \ definitions$
process $User =$
W : call$^\star[\top]\langle$this.loc$\rangle.W$
+
take$[$loc $==$ this.loc$]\langle$this.dest$\rangle$.**kill**
endprocess

process $Taxi =$
F : take$[\top](d)\{$dest $\leftarrow d$, occupied $\leftarrow 1\}.G$
+
call$^\star[$this.loc $\neq$ loc$](d)\{$dest $\leftarrow d\}.G$
G : move$^\star[\bot]\langle\circ\rangle\{$loc $\leftarrow$ dest, dest $\leftarrow \bot$, occupied $\leftarrow 0\}.F$
endprocess

process $Arrivals =$
A : arrival$^\star[\bot]\langle\circ\rangle.A$
endprocess

$//Environment\ \ definitions$
$\#(Taxi[F], \text{loc} == \ell)$

prob{
$\top$, take $\rightarrow \frac{1}{\#(Taxi[F], \text{loc}==\text{sender.loc})}$
$\top$, call$^\star \rightarrow p_{lost}$
default 1
}

rate{
$\top$, take $\rightarrow \lambda_t$
$\top$, call$^\star \rightarrow \lambda_c$
$\top$, move$^\star \rightarrow mtime(\text{now}, \text{sender.loc}, \text{sender.dest})$
$\top$, arrival$^\star \rightarrow atime(\text{now}, \text{loc})$
default 0
}

update{
$\top$, arrival$^\star \rightarrow \{\ \}, (User[W], \{\text{loc} = \text{sender.loc}, \text{dest} = destLoc(\text{now}, \text{sender.loc}))$
}

measure waitingAt$_\ell$ $= \#(User[\star], \text{loc} == \ell)$

function $mtime(t, l_1, l_2) = \lambda_{step} \cdot \max\{|\pi_1(l_1) - \pi_1(l_2)| + |\pi_2(l_1) - \pi_2(l_2)|, 1\}$;
function $atime(time, l) = \lambda_a \cdot \frac{1}{|L|}$;
function $destLoc(time, l) =$ if $(l == (1,1))$ then $rand(L - l)$ else $(1,1)$;

collective{
$(Taxi[F], \text{loc} = (0,0)$, dest $= \bot$, occupied $= 0) \parallel$
$\cdots \parallel (Taxi[F], \text{loc} = (i,j)$, dest $= \bot$, occupied $= 0) \parallel$
$\cdots \parallel (Taxi[F], \text{loc} = (2,2)$, dest $= \bot$, occupied $= 0) \parallel$
$(Arrval[A], \text{loc} = (0,0)) \parallel \cdots \parallel (Arrval[A], \text{loc} = (2,2))\}$

Fig. 1. The CARMA model of the Smart Taxi System

of this approach is that one can evaluate the same behaviour (collective) under different models for the enclosing environment.

In this section we consider a scenario with a grid of 3×3 patches, a set of locations (i, j) where $0 \leq i, j \leq 2$, and we instantiate the environment of the *smart taxi system* with respect to two different specifications for the environment:

Scenario 1: Users arrive in all the patches at the same rate;

Scenario 2: At the beginning users arrive with a higher probability to the patches at the border of the grid; subsequently, users arrive with higher probability in the centre of the grid.

In both the scenarios users in the border will use the taxi to go to the *centre*, while users collected in the centre will use the taxi to go to any other location (the destination is probabilistically selected). In both the scenarios we assume that the movement rate is constant and is proportional to the number of patches to be traversed to reach the destination. In both the considered scenarios the collective has the structure reported below:

$$
\begin{array}{l}
(Taxi[F], \mathsf{loc} = (0,0)\ ,\ \mathsf{dest} = \perp\ ,\ \mathsf{occupied} = 0) \parallel \\
\cdots \parallel (Taxi[F], \mathsf{loc} = (i,j)\ ,\ \mathsf{dest} = \perp\ ,\ \mathsf{occupied} = 0) \parallel \\
\cdots \parallel (Taxi[F], \mathsf{loc} = (2,2)\ ,\ \mathsf{dest} = \perp\ ,\ \mathsf{occupied} = 0) \parallel \\
(Arrivals[A], \mathsf{loc} = (0,0)) \parallel \cdots \parallel (Arrivals[A], \mathsf{loc} = (2,2))
\end{array}
$$

Above we consider $k = 5$ taxis in each location.

To instantiate the environment for **Scenario 1** considered above, we have just to instantiate functions *mtime* and *atime* of Example 5 and function *destLoc* of Example 6:

$$
\begin{array}{l}
\mathsf{function}\ mtime(t, l_1, l_2) = \lambda_{step} \cdot \max\{|\pi_1(l_1) - \pi_1(l_2)| + |\pi_2(l_1) - \pi_2(l_2)|, 1\}; \\
\mathsf{function}\ atime(t, l) = \lambda_a \cdot \frac{1}{|L|}; \\
\mathsf{function}\ destLoc(t, l) = \mathsf{if}\ (l == (1,1))\ \mathsf{then}\ rand(L - l)\ \mathsf{else}\ (1,1);
\end{array}
$$

Above λ_{step} is the rate needed to move from one location to an adjacent one, π_1 (resp. π_2) is a function that returns the first (resp. the second) element of a pair, λ_a is the arrival rate of a user in the system,L is the set of the considered localities while *rand* is used to select randomly an element in the set received as parameter. The complete CARMA specification of our system is presented in Figure 1. The results of the simulation of the resulting CARMA model is reported in Figure 2. In the left side of the figure we can observe the average number of users that are waiting for a taxi in the location $(1,1)$ and in one location in the border of the grid, namely $(0,0)$[5]. In the right hand side of the same figure we can present the proportion of free taxis that are waiting for a user at location $(1,1)$ and $(0,0)$, respectively, and the fraction of taxis that are moving from one patch to another without a user (these are the taxis that are relocating after a *call* has been received). The remaining taxis (not shown) are engaged by users.

[5] Due to the symmetry of the considered model, any other location in the border presents similar results.

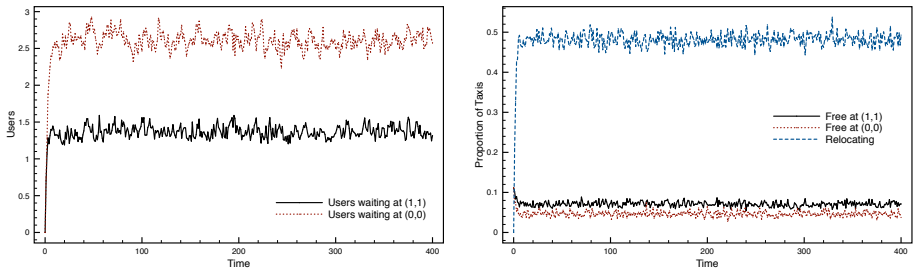

Fig. 2. Smart Taxi System: Scenario 1

We can notice that, on average and after an initial startup period, around 2.5 users are waiting for a taxi in the location in the periphery of the grid while only 1.5 users are waiting for a taxi in location $(1,1)$. This is due to the fact that in **Scenario 1** a larger fraction of users are delivered to location $(1,1)$, that is the central patch. For this reason, a larger fraction of taxis will soon be available to collect users at the centre whereas to collect a user from the border, a taxi has to change its location. This aspect is also witnessed by the fact that, in this scenario, a large fraction of taxis (around 50%) are continually moving between the different patches.

The simulation of **Scenario 2** is reported in Figure 3. The environment for this scenario is exactly the same as considered for the previous one except for function *atime*. Now this function takes into account the current time (parameter now) to model the fact that the arrival of new users depends on current time, just as we might expect traffic patterns within a city to vary according to the time of day. We assume that from time 0 to time 200, $3/4$ of users *arrive* on the border while only $1/4$ request a taxi in the city centre. After time 200 these values are switched. New definition of function *atime* is the following:

$$\begin{aligned}\mathsf{function}\ \mathit{atime}(t,l) = \mathsf{if}\ l == (1,1)\ \mathsf{then}\ & \\ & \mathsf{if}\ t < 200\ \mathsf{then}\ \tfrac{1}{4}\cdot\lambda_a\ \mathsf{else}\ \tfrac{3}{4}\cdot\lambda_a \\ \mathsf{else}\ & \\ & \mathsf{if}\ t < 200\ \mathsf{then}\ \tfrac{3}{4}\cdot\lambda_a\ \mathsf{else}\ \tfrac{1}{4}\cdot\lambda_a;\end{aligned}$$

We can notice that the results obtained from time 0 to time 200 are similar to the ones already presented for the first scenario. However, after time 200, as expected, the number of users waiting for a taxi in the border decreases below 1 whilst the average waiting for a taxi in the centre increases to just over 1. Since after time 200 a large proportion of users request a taxi in the centre, the fraction of taxis that change their location without a user decreases from 40% to 20%.

In both the scenarios one can observe that even if only a small number of users are waiting for a taxi, a significant fraction of taxis are continually moving from one patch to another without users (i.e. in a free state). This is mainly due to the fact that the action used to *call* a taxi is a broadcast output. As a consequence we have that even if only a single user needs a taxi at a given location, all the *free* taxis can change their position to satisfy this request. To study this aspect in more detail, we consider now a variant of

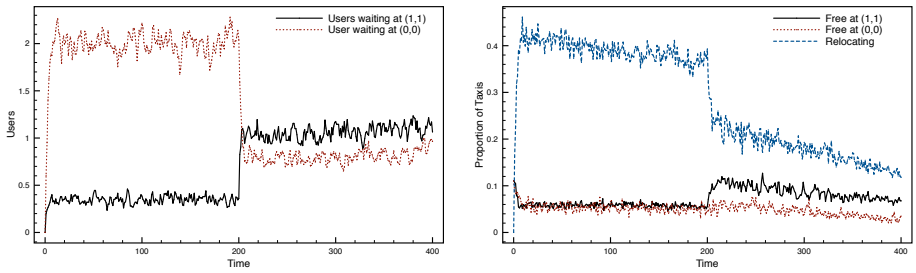

Fig. 3. Smart Taxi System: Scenario 2

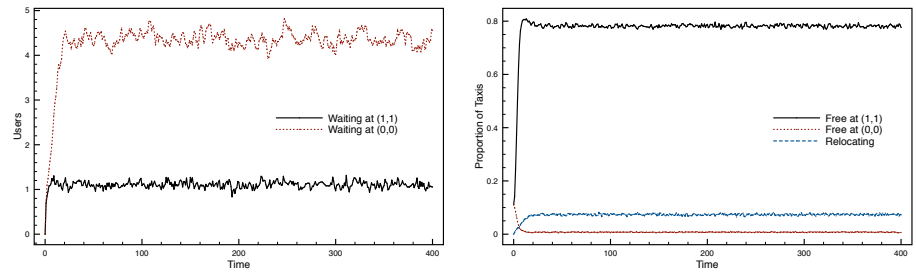

Fig. 4. Smart Taxi System: Scenario 1 (modified specification)

processes *Taxi* and *User* where action call is no longer a broadcast output, but is instead a unicast output. The CARMA representation of the variants of these two processes is reported below:

$$
\begin{array}{l}
\mathsf{process}\ \mathit{User} = \\
\quad W : \mathsf{call}[\top]\langle \mathsf{this.loc}\rangle.W \\
\qquad + \mathsf{take}[\mathsf{loc} == \mathsf{this.loc}]\langle \mathsf{this.dest}\rangle.\mathbf{kill} \\
\mathsf{endprocess}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{process}\ \mathit{Taxi} = \\
\quad F : \mathsf{take}[\top](d)\{\mathsf{dest} \leftarrow d, \mathsf{occupied} \leftarrow 1\}.G \\
\qquad + \mathsf{call}^{\star}[\mathsf{this.loc} \neq \mathsf{loc}](d)\{\mathsf{dest} \leftarrow d\}.G \\
\quad G : \mathsf{move}^{\star}[\bot]\langle\circ\rangle\{\mathsf{loc} \leftarrow \mathsf{dest}, \mathsf{dest} \leftarrow \bot, \mathsf{occupied} \leftarrow 0\}.F \\
\mathsf{endprocess}
\end{array}
$$

The results of simulating the two scenarios with the modified specifications are reported in Figure 4 and Figure 5. In both cases we can observe that the number of users waiting for a taxi in patches located in the border of the grid doubles, whilst almost all taxis will wait for users' calls in the centre location (around 80%). This means that after an initial startup period all the taxis will be always staying in the central location and the patch arrangement of the city is, in fact, no longer used in the model.

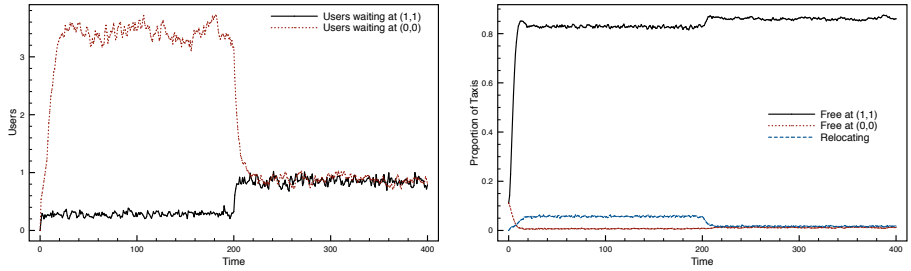

Fig. 5. Smart Taxi System: Scenario 2 (modified specification)

5 Modelling the environment

A key feature of CARMA is its distinct treatment of an *environment*. It should be stressed that although this is an entity within our model, it is intended to represent something more pervasive and diffuse in the real system, which is abstracted within the modelling to be an entity which exercises influence and imposes constraints on the more physical entities in the system, be they software, human or other forms of agents. For example, in the Smart Taxi System modelled in CARMA the environment determines the rate at which taxis may move through the city, an abstraction of the presence of other vehicles causing congestion which may impede the progress of the taxi to a greater to lesser extent at different times of the day. Thus it should not be considered that the presence of an environment in the model implies the existence of centralised control in the system.

This view of the environment coincides with the view taken by many within the situated multi-agent community e.g. [1]. Specifically, in [2] Weyns *et al.* argue the importance of having a distinct environment within every multi-agent system. Whilst they are viewing such systems from the perspective of software engineers, many of their arguments are as valid when it comes to modelling a multi-agent or collective adaptive system as they are when it comes to building one. Thus our work can be viewed as broadly fitting within the same framework, albeit with a higher level of abstraction. Just as in the construction of a system, in the construction of a model distinguishing clearly between the responsibilities of the agents and of the environment provides separation of concerns and assists in the management of inevitably complex systems. In [2] the authors provide the following definition:

> "The environment is a first-class abstraction that proves the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources."

This is the role that the environment plays within CARMA models through the evolution rules. However, in contrast to the framework of Weyns *et al.* the environment in a CARMA model is not an active entity in the same sense as the agents are active entities; in our case the the environment is constrained to work *through* the agents, by influencing their dynamic behaviour or by changes to the number and types of agents making

up the system. Despite this difference we consider the CARMA conceptual model to be in line with the Weyns *et al*. framework, the difference primarily arising due to the more abstract approach needed for modelling rather than implementing a CAS. Moreover, note that in CARMA the environment is formally specified and integrated into the operational semantics of the language.

In [24], Saunier *et al*. advocate the use of an *active environment* to mediate the interactions between agents, an active environment that is aware of the current context for each agent. The environment in CARMA also supports this view, as the evolution rules in the environment take into account the state of all the potentially participating components to determine both the rate and the probability of communications being successful, thus achieving a multicast not based on the address of the receiving agents as suggested by Saunier *et al*. This is what we term attribute-based communication in CARMA. Moreover, when the application calls for a centralised information portal, the global store in CARMA can represent it. The higher level of abstraction used in CARMA means that many implementation issues are elided but the CARMA environment could be viewed as capturing the EASI (Environment for Active Support of Interaction) environment of Saunier *et al.* [24], although in CARMA the *filter* is more closely associated with the actions. However, just as in EASI, filters (predicates) may specified separately by the sender, the receiver and the environment. However, our predicates are more strict and "overhearing" type interactions must be anticipated by the modeller, since the effect is taken to be the conjunction of the sender, receiver and environment predicates, thus removing the need for policies to arbitrate between conflicting filters.

The role of the environment is related to the spatially distributed nature of CAS — we expect that *where* an agent is will have an effect on *what* an agent can do. Thus we do find similar features in modelling languages targeted at other domains where location is considered to have influence on the possible behaviours of agents. For example several formalisms developed in the context of biological processes, especially intracellular processes, capture the spatial arrangement of elements in the system because this can have a profound effect on the behaviours that can be observed. In this context, the most important aspect of the spatial arrangement is often hierarchical and logical, rather than the actual physical placement of elements, this is sometimes termed *multi-level* modelling. Moreover the concept of level may also refer to organisational levels, as well as physical levels so that the relationship between levels might be characterised as *consists of* or *contains* [25]. Here we are particularly concerned with modelling techniques that seek to faithfully represent the stochastic dynamic behaviour of systems, allowing properties such as performance, availability and dependability to be assessed.

One example of such a language is the ML-RULES formalism developed by Maus *et al* [26]. Here the focus is on hierarchical nesting of biological entities and the underlying semantics is given in terms of a population CTMC, intended for analysis by simulation. Entities can be created "on demand" but this must be programmed within the underlying simulation engine by the modeller, and is not supported at the level of the modelling language itself. Rules are used to define the possible reactions in the system and the state updates which result from them; rules are applied by pattern-matching. As in CARMA agents are equipped with attributes and these may be used to filter the rules which may be applied, although there is no explicit naming of attributes making it difficult

for other agents to access current values. Moreover it is the modeller's responsibility to ensure that attributes are used consistently across different rules. Explicit function calls are made within agents to determine execution parameters such as Markovian rates or probabilities, thus assuming agents have direct access to a global knowledge. In contrast, in CARMA it is assumed that access to global knowledge is restricted to the environment. Allowing agents to directly access global knowledge makes it more difficult to consider the same set of agents in a different context, because there is less clear separation between the agents and their environment. The multi-level aspect of ML-RULES is used to capture a form of vertical causation, where the application of a rule at one level triggers an update within another level of the model.

The upward and downward causation are also key features of the ML-DEVS formalism, presented in [27], although in a slightly more restricted form since here agents can only trigger changes in the levels immediately above or below them, whereas in ML-Rules an impact can be across arbitrary levels in the hierarchy. ML-DEVS is a modular, hierarchical formalism which is intended to represent a reactive system which interacts asynchronously with its "environment". However the formalism does not support the notion of a distinguished environment, as in CARMA, but rather considers the environment of an agent to be the agents are the same level, with which it may form horizontal couplings, and those in the adjacent levels, with which it may form vertical couplings.

BioSpace [28] is a novel process calculus designed for modelling the physical arrangement of biological molecules in applications such as the formation of polymers and the interactions between microbes and biomaterials [29]. Individual agents in the calculus represent the biological entities, but operate in a type environment against which the legitimacy of their actions can be checked: each action and the entities it involves have types and type consistency is checked before the model evolves. BioSpace^L extends BioSpace by allowing the explicit placement of entities, and giving the modeller the power to program location updates. This reflects the key role that location plays in the considered biological applications, but the physical environment is represented rather implicitly.

In the formalism presented in [25], the formalism supports a multi-level approach which is based on organisational rather than spatial structures. Each level consists of a number of agents whose behaviour may depend on agents at a lower level. In this *system of systems*, agents are represented by automata and automata are organised in tree-structures representing how agents are one level are constituted from their child automata. For example, in a biological setting, the top level might be *tissue* which may alternate between healthy and diseased states; this may be made up by *cells* and the state of the cells within the tissue will influence its health or otherwise; the behaviour and state of cells will depend on the biochemical networks within them, themselves made up of proteins in various states of abundance. Again there are notions of horizontal and vertical couplings, and agents which are above provide the environment to those that are below, in a hierarchical manner.

Similarly the ambient calculus [30], and its biological dialect, bio-ambients [31], capture the behaviour of elements within a system, with respect to a hierarchical arrangement of physical or logical space. As elements move into or out of domains, their

behaviour may change because they change their context of operation and communication is limited to be local.

In contrast to these multi-level models, in CARMA we restrict to two levels. The behaviour of the entities within the system are captured by the collective, and this is placed in the context of an environment, which is distinct from any entity and which has the power to constrain the behaviour of the entities through the evolution rule. This reflects our treatment of location in terms of physical space, rather than the hierarchical arrangement of space commonly used in biological modelling where the emphasis is on the containment relationship. It is noteworthy that BioSpace^L, which has similar focus of the physical rather than logical representation on space, also takes a two level approach with the entities and the environment. Within the collective there is no hierarchy, although a single component may have behaviour comprised of multiple process automata. This is intended to represent different aspects of the behaviour of a single entity, such as the taxi having two processes F and G capturing the orthogonal aspects of its behaviour, relating to passengers and movement in search of a passenger respectively.

A two level approach is also found in quantitative formalisms such as PEPA nets [32], Spatial PEPA [33] and STOKLAIM [34]. These languages were motivated by mobile computing and therefore share the aim from modelling CAS to capture systems with behaviour distributed over physical space, in which the current location or position of an entity, and in particular the entities that are co-located, may alter the actions that can be undertaken. In PEPA nets and Spatial PEPA a graphical representation of the physical locations is used either as a Petri net, or as a hyper-graph, and the physical structure is taken to be static. In STOKLAIM, in contrast, the processes within the system may explicitly control the physical structure of the global network. Despite some success in modelling the mobile computing scenarios of the time, these languages are not equipped to represent large populations of entities with similar behaviour, thus they are not well-suited to capture the collective nature of CAS. This large scale nature of CAS systems makes it essential to support scalable analysis techniques, thus CARMA has been designed anticipating both a discrete and a continuous semantics in the style of [18].

6 Conclusions

As we begin to see the deployment of collective adaptive systems in modern infrastructures, it is essential that we have tools and techniques to analyse the behaviour of systems comprised of multiple populations of agents both in terms of their functional and non-functional requirements. In this paper we have introduced CARMA, a novel modelling language which aims to represent collectives of agents working in a specified environment and support the analysis of quantitative aspects of their behaviour such as performance, availability and dependability. CARMA is a stochastic process algebra-based language combining several innovative features such as a separation of behaviour and knowledge, locally synchronous and globally asynchronous communication, attribute-defined interaction and a distinct environment which can be changed independently of the agents. We have demonstrated the use of CARMA on a smart taxi

system example, showing the ease with which the same system can be studied under different contexts or environments.

The quantified nature of CARMA, based on Markovian agents whose actions have probability and duration governed by an exponentially distributed random variable, means that CARMA models are amenable to analysis by a variety of techniques. In this paper, based on the semantics presented in [3], we have studied the behaviour of the smart taxi system via stochastic simulation. Future work involves extending the tool support for CARMA and further developing the semantics to also encompass scalable analysis via mean field approximation in the style of [18]. Future work will also include more investigation of the use of the environment to study the adaptivity of systems.

References

1. Danny Weyns and Tom Holvoet. A formal model for situated multi-agent systems. *Fundam. Inform.*, 63(2-3):125–158, 2004.
2. Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, 2007.
3. L.Bortolussi, R. De Nicola, V.Galpin, S.Gilmore, J.Hillston, D.Latella, M.Massink, and M.Loreti. Carma: Collective adaptive resource-sharing markovian agents. In *Proc. of the Workshop on Quantitative Analysis of Programming Languages 2015*, page to appear. Springer, 2015.
4. Jane Hillston. *A Compositional Approach to Performance Modelling*. CUP, 1995.
5. Holger Hermanns and Michael Rettelbach. Syntax, Semantics, Equivalences and Axioms for MTIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.
6. Marco Bernardo and Roberto Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Theoretical Computer Science*, 202(1-2):1–54, 1998.
7. Corrado Priami. Stochastic π-calculus. *The Computer Journal*, 38(7):578–589, 1995.
8. Federica Ciocchetta and Jane Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, 410(33):3065–3084, 2009.
9. Henrik C. Bohnenkamp, Pedro R. D'Argenio, Holger Hermanns, and Joost-Pieter Katoen. MODEST: A compositional modeling formalism for hard and softly timed systems. *IEEE Trans. Software Eng.*, 32(10):812–830, 2006.
10. Holger Hermanns, Ulrich Herzog, and Joost-Pieter Katoen. Process algebra for performance evaluation. *Theor. Comput. Sci.*, 274(1-2):43–87, 2002.
11. Luca Bortolussi and Alberto Policriti. Hybrid dynamics of stochastic programs. *Theor. Comput. Sci.*, 411(20):2052–2077, 2010.
12. Rocco De Nicola, Michele Loreti, Rosario Pugliese, and Francesco Tiezzi. A formal approach to autonomic systems programming: The SCEL language. *TAAS*, 9(2):7, 2014.
13. Cheng Feng and Jane Hillston. PALOMA: A process algebra for located markovian agents. In *Quantitative Evaluation of Systems - 11th International Conference, QEST 2014, Florence, Italy, September 8-10, 2014. Proceedings*, volume 8657 of *Lecture Notes in Computer Science*, pages 265–280. Springer, 2014.
14. M.John, C.Lhoussaine, J.Niehren, and A.M.Uhrmacher. The attributed Pi calculus. In *Proc. of Computational Methods in Systems Biology*, volume 5307 of *LNBI*, pages 83–102, 2008.
15. Diego Latella, Michele Loreti, Mieke Massink, and Valerio Senni. Stochastically timed predicate-based communication primitives for autonomic computing. In Nathalie Bertrand

and Luca Bortolussi, editors, *Proceedings Twelfth International Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2014, Grenoble, France, 12-13 April 2014.*, volume 154 of *EPTCS*, pages 1–16, 2014.
16. Yehia Abd Alrahman, Rocco De Nicola, Michele Loreti, Francesco Tiezzi, and Roberto Vigo. A calculus for attribute-based communication. In *Proceedings of SAC 2015*, 2015. To appear.
17. Cheng Feng and Jane Hillston. Speed-up of stochastic simulation of PCTMC models by statistical model reduction, 2015. Submitted.
18. Mirco Tribastone, Stephen Gilmore, and Jane Hillston. Scalable differential analysis of process algebra models. *IEEE Transactions on Software Engineering*, 38(1):205–219, 2012.
19. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.
20. Alexander Helleboogh, Giuseppe Vizzari, Adelinde M. Uhrmacher, and Fabien Michel. Modeling dynamic environments in multi-agent simulation. *Autonomous Agents and Multi-Agent Systems*, 14(1):87–116, 2007.
21. Rafael H. Bordini, Fabio Y. Okuyama, Denise de Oliveira, Guilherme Drehmer, and Romulo C. Krafta. The MAS-SOC approach to multi-agent based simulation. In *Regulated Agent-Based Social Systems, First International Workshop, RASTA 2002, Bologna, Italy, July 16, 2002, Revised Selected and Invited Papers*, volume 2934 of *Lecture Notes in Computer Science*, pages 70–91. Springer, 2004.
22. R. Zalila Mili and Renee Steiner. Modeling agent-environment interactions in adaptive MAS. In *Engineering Environment-Mediated Multi-Agent Systems, International Workshop, EEMMAS 2007, Dresden, Germany, October 5, 2007. Selected Revised and Invited Papers*, volume 5049 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 2008.
23. Jose R. Celaya, Alan A. Desrochers, and Robert J. Graves. Modeling and analysis of multi-agent systems using petri nets. *JCP*, 4(10):981–996, 2009.
24. Julien Saunier, Flavien Balbo, and Suzanne Pinson. A formal model of communication and context awareness in multiagent systems. *Journal of Logic, Language and Information*, 23(2):219–247, 2014.
25. Luca Bortolussi, Jane Hillston, and Mirco Tribastone. Fluid performability analysis of nested automata models. *Electr. Notes Theor. Comput. Sci.*, 310:27–47, 2015.
26. C.Maus, S.Rybacki, and A.M.Uhrmacher. Rule-based multi-level modelling of cell biological systems. *BMC Systems Biology*, 5:166, 2011.
27. A.Steiniger, F.Krü ger, and A.M.Uhrmacher. Modeling agents and their environment in Multi-Level-DEVS. In *Proc. of the 2012 Winter Simulation Conference*, Berlin, Germany, 2012. IEEE.
28. A.Compagnoni, P.Giannini, C.Kim, M.Milideo, and V.Sharma. A calculus of located entities. In *Proc. of DCM 2013*. ArXiv, 2014.
29. A.Compagnoni, V.Sharma, Y.Bao, M.Libera, S.Suhkishvili, P.Bidinger, L.Bioglio, and E.Bonelli. Biospace: A modeling and simulation language for bacteria-materials interactions. *ENTCS*, 293:35–49, 2013.
30. Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.
31. Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Y. Shapiro. Bioambients: an abstraction for biological compartments. *Theor. Comput. Sci.*, 325(1):141–167, 2004.
32. Stephen Gilmore, Jane Hillston, Leila Kloul, and Marina Ribaudo. PEPA nets: a structured performance modelling formalism. *Performance Evaluation*, 54:79–104, 2003.
33. Vashti Galpin. Modelling network performance with a spatial stochastic process algebra. In *Proc. of International Conference on Advanced Information Networking and Applications*, pages 41–49. IEEE, 2009.

34. Rocco De Nicola, Diego Latella, and Massink Massink. Formal modeling and quantitative analysis of klaim-based mobile systems. In *Proc. of the 2005 ACM Symposium on Applied Computing (SAC), Santa Fe, New Mexico, USA, March 13-17, 2005*, pages 428–435. ACM, 2005.